Aalto University
School of Science
Machine learning, Data science and AI (CCIS)

Ossi Arasalo

Human pose estimation from depth images

Master's Thesis
Espoo, October 29, 2019

Supervisor: Assistant Professor Perttu Hämäläinen
Advisor: Otto Korkalo M.Sc. (Tech.)

Aalto University
School of Science
Machine learning, Data science and AI (CCIS)

ABSTRACT OF
MASTER'S THESIS

Author:	Ossi Arasalo		
Title:	Human pose estimation from depth images		
Date:	October 29, 2019	**Pages:**	52
Major:	Computer Science	**Code:**	SCI3044
Supervisor:	Assistant Professor Perttu Hämäläinen		
Advisor:	Otto Korkalo M.Sc. (Tech.)		

Human pose estimation has many applications from activity analysis to autonomus cars. Modern advances in deep learning research have enabled real time multi-person pose estimation in complex environments. In this thesis, a state of the art deep learning architecture is adapted to work with depth sensors. A dataset is generated using computer graphics instead of annotating thousands of images by hand. Results are promising; trained neural network detects humans in multi-person environments even when occlusion is present. However, there are challenges rising from the difference between the real world and the synthetic data generation, which has to be addressed.

Keywords:	deep neural network, articulated pose estimation, depth imaging, machine learning, domain randomization
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Koneoppiminen, data-analyysi, tekoäly (CCIS)

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Ossi Arasalo		
Työn nimi:	Ihmisten asennon tunnistus syvyyskameralla		
Päiväys:	29. lokakuu 2019	**Sivumäärä:**	52
Pääaine:	Tietotekniikka	**Koodi:**	SCI3044
Valvoja:	Apulaisprofessori Perttu Hämäläinen		
Ohjaaja:	Diplomi-insinööri Otto Korkalo		

Automaattisella ihmisten asennon tunnistuksella on lukuisia sovelluksia aktiviteettianalyysistä itsenäisiin autoihin. Nykyaikainen kehitys syvien neuroverkkojen alalla on mahdollistanut useamman ihmisen reaaliaikaisen asennon tunnistamisen monimutkaisissa ympäristöissä. Tässä diplomityössä adaptoidaan nykyaikainen neuroverkko arkkitehtuuri toimimaan syvyyskameralla saaduilla kuvilla. Neuroverkon opetukseen tarvittava opetusdata generoidoon tietokonegrafiikan avulla sen sijaan, että opetusdata luotaisiin käsityönä. Saadut tulokset ovat lupaavia; opetettu neuroverkko kykenee tunnistamaan samanaikaisesti usean ihmisen monimutkaisessa ympäristössä. Kaikesta huolimatta, simuloidun datan ja todellisen maailman välinen eroavaisuus aiheuttaa ongelmia, jotka täytyy ottaa huomioon.

Asiasanat:	syvät neuroverkot, asennon tunnistus, syvyyskamera, koneoppiminen
Kieli:	Englanti

Acknowledgements

Thanks to Otto Korkalo, Paul Kemppi and Perttu Hämäläinen for all the help with the thesis process.

Abbreviations and Acronyms

PAF	Part affinity field
ANN	Artificial neural network
CNN	Convolutional neural network
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution
$\mathcal{U}(a, b)$	Discrete uniform distribution
$U(a, b)$	Continuous uniform distribution
PCKh	Percentage of correct keypoints
OKS	Object keypoint similarity

Contents

Chapter 1

Introduction

Automatic human pose detection has many applications ranging from human-computer interaction and surveillance to autonomous vehicles and healthcare. Finding the skeleton transforms the high dimensional image data into a lower dimensional representation that contains the most relevant information to do further analysis. The task has obviously attracted a lot of attention in computer vision research due to a huge commercial potential. However, localizing human joints in occluded multi-person scenarios with great accuracy is a daunting task. Many traditional computer vision algorithms fail when the situation is dynamic and not easily constrained.

Recent advances in computing hardware performance and deep neural network research have provided novel approaches to solve the problem. Architectures such as OpenPose and DeepPose have shown incredible performance in difficult use cases. Neural networks can be trained with minimal constraints about the scenario, which makes them more robust to varying situations. Nonetheless, these architectures require huge amounts of training data to give reasonable performance. The real difficulty is collecting a representative dataset that covers all possible real world situations thoroughly.

Nearly all artificial neural network (ANN) architectures are trained to recognize humans from RGB images. Color cameras are everywhere, which makes them the obvious first choice for 2D human pose estimation as well. However, in certain use cases, depth cameras are the better choice. Depth sensors are privacy preserving and relatively invariant to different lighting conditions. Moreover, depth cameras give actual 3D information of the scenario, which opens new possibilities for 3D pose estimation.

Perhaps the greatest advantage of using depth cameras is the possibility to simulate the training data using computer graphics. Depth images are textureless and quite unaffected to lighting changes, which makes them more approachable to simulation then RGB counterpart. Furthermore, manual

annotation process can be avoided and the quality of the labels can be guaranteed. Hence, the most time consuming and expensive part of the machine learning pipeline can be avoided.

1.1 Objectives

The objective of this thesis is to implement a 2D pose estimation pipeline that can predict human skeleton of multiple persons simultaneously from a single depth camera image. Deep learning research has mainly focused on RGB based pose estimation, and has been used in depth imaging only a few times.

First part of this implementation is to create a data simulator with computer graphics techniques, that generates images that are varied to a high degree so the real world is modeled as well. These images are then used to train a modified deep learning pose estimator architecture called OpenPose to generate pose estimates. An existing implementation is used as basis for modifications [10].

The goal of this thesis is not to train a network that works in all situation. It is difficult to generate training data that represents all possible human poses and different environment settings. Therefore, use cases are restricted to two different settings. The first use case is to train a network, that predicts poses as well as possible in a meeting room setting. The camera is in a similar position as a security camera. The setting will mostly consist of people sitting around the table and walking to the door. The generated synthetic dataset should reflect this as well as possible. The second use case is detecting the upper body of a truck driver while he/she is driving the vehicle. This setup is extremely simple, because there can be only one person in the image and the position of the driver is restricted between 50-150 cm. Furthermore, occlusion happens rarely.

Experiments are run with different preprocessing techniques to compare how they affect the performance in both use cases. These include different depth image colorings, simulation setups and network designs. Image coloring has been studied previously in image classification, but not in pose estimation.

1.2 Structure of the thesis

The next chapter background 2, covers the current state of pose estimation focusing on depth sensor based approaches. Additionally, recent advances in

deep learning pose estimation is covered in more detail. In chapter methods 3, the implementation choices are discussed. Chapter 4 explains the experiment setup, and chapter results 5, presents the performance of different networks. Chapter discussion 6, examines the findings. Finally, chapter conclusions 7, sums up the results of this thesis.

Chapter 2

Background

2.1 Human pose estimation from RGB images

Human pose estimation is the process of automatic detection of the skeletal pose in a single or multi-person environment. The process consists of finding the keypoints, such as elbows and shoulders, and combining these into a skeleton model that describes the kinematic structure of a human.

The earliest approaches to the human pose estimation used markers that would be attached to the subject. Markers are easy to distinguish from the surroundings, so the detection is simple. These detections can then be transformed into a full skeleton using a kinematic model. Marker based approaches are reliable and accurate, but are quite impractical, because the markers need to be worn [26].

Finding the keypoints automatically without the need to wear any specialized markers was the next major advancement in human pose estimation [26]. Automated classification is done by computing feature descriptors from images and labeling these to different body parts. Common image descriptors are a combination of silhouettes, edges and colors [34].

An example of automated classification is template matching, which has shown great success in human pose estimation. One of the most influential papers to template matching is the pictorial structure modeling [16]. Each body part of a human skeleton is modeled separately and these parts are joined together by flexible connections [16]. This model can be understood as an undirected graph (V, E), where the vertices are the body parts and edges are the joints. The matching of this pictorial representation can be formulated as an energy minimization problem where different body parts are parameterized by pixel locations. The choice for the optimization function is difficult, because the joints are geometrically constrained. Therefore, the

function has to model all possible human poses as accurately as possible.

Proposed methods can be roughly divided into two approaches. In top-down approach, human models are matched directly to the image [34]. This means that the pose estimates are iteratively improved after a rough initial guess. These approaches are prone to problems with occlusion and high computational cost [34]. The other approach called bottom-up, works by first finding the individual keypoint locations and applying the kinematic model afterwards [34]. These models handle person occlusion better, but the keypoint matching problem is computationally heavy in multi-person setting. Another drawback is false positive keypoint detections that complicate the kinematic model matching process.

2.2 Human pose estimation from depth images

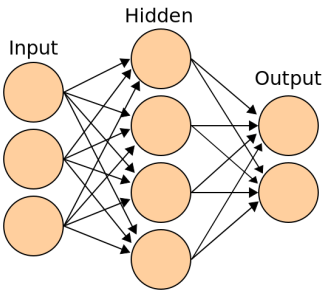
The same techniques that are used in RGB based pose estimation are utilized in the depth domain as well. Some preprocessing steps such as background subtraction are easier with depth images, so they are often used to simplify the problem [11].

One widely known depth based pose estimation algorithm was introduced by Microsoft Research [39]. Their solution first calculates depth invariant features by normalizing each pixel and then applying randomized decision forests to label each of these features into a body part [39]. Joint proposals are then found by applying mean shift algorithm with Gaussian kernel, which can be connected into a full skeleton [39]. This approach was further improved by using a constrained kinematic model called Vitruvian manifold, that helps to address unfeasible poses [42].

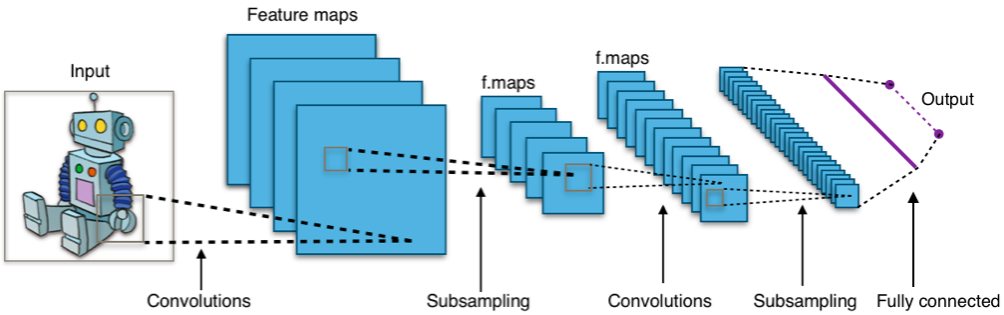
Human pose can be estimated from 3D point cloud as well [22]. Their approach uses ICP algorithm to match a human body model to a point cloud with a separate nonlinear optimization to match each body segment to the right place [22].

2.3 Deep learning

Artificial (deep) neural network (ANN) is a machine learning method that learns to model complex problems by combining simple nonlinear functions on top of each other [21]. Classic machine learning algorithms learn the mapping from the representation to output, which often requires feature engineering for the inputs. In turn, deep neural networks learn the representation as well, which means that the algorithm automatically finds the most



(a) Feedforward neural network



(b) Convolutional neural network

Figure 2.1: Example deep neural network architectures

important features [21].

The simplest example of a deep learning algorithm is a multilayer perceptron. It consists of an input layer, n hidden layers and an output layer. Each neuron is connected to all neurons in the next layer. Nonlinearity is achieved by applying a nonlinear function to the outputs of each neuron [21]. Example of a deep feedforward network can be seen in Figure 2.1(a).

Convolutional neural network (CNN) is a specialized neural network that is designed for grid data [21]. CNNs have been used with a lot of success in computer vision applications. As the name suggests, CNNs use convolution operation instead of matrix multiplication in some parts of the network [21]. This allows reducing the number of parameters compared to the feedforward network, and still learn representative features. An example architecture can be seen in Figure 2.1(b). The first layers of the network learn to detect simple features such as edges. The deeper the network gets, the more abstract features it can represent. CNNs are nearly always connected to a shallow feedforward network to do classification or regression.

2.4 Deep learning based pose estimation

Convolutional neural networks have achieved state-of-the-art results in many different areas due to their robustness and huge learning capacity. Many traditional approaches fail in dynamic environments where the background might be changing over time. A case in point, pose estimation algorithm used in Microsoft Kinect sensor does background subtraction to extract a person from the image [39]. Therefore, if the camera moves while the prediction algorithm is running, the background changes and pose estimation fails.

Multi-person deep learning methods can be divided roughly into two categories, top-down and bottom-up solutions. Both approaches have shown a lot of interest in recent research.

In top-down approach, the pose is estimated by first running a person detector to find every person from the image and then running a single pose estimator for each of these extracted persons. The performance of these algorithms is heavily relying on accurate person detection. This is intuitive, because the bounding box has to contain all keypoints for a single person, and if there is a mistake, the results will be inaccurate. The runtime is proportional to the number of persons detected by the detector part of the network. There are multiple successful solutions using this approach [15, 25, 41].

On the other hand, in bottom-up pose estimation the procedure is the opposite. First, keypoints and joint positions are estimated, and at the end these joint candidates are associated with individual persons. Part association is a NP-hard graph matching problem. Keypoints and joint estimates create a fully connected graph that needs to be divided into smaller subgraphs. These subgraphs are the individual persons that the goal is to estimate. Bottom-up approach is computationally extremely heavy due to the challenging graph problem. However, there are a few greedy approximation algorithms that generate good results while achieving real time performance such as the OpenPose integer linear programming approximation [8].

2.4.1 OpenPose

One of the best known and best performing pose estimation architecture for RGB images is the OpenPose developed at the CMU Perceptual Computing Lab[7, 8]. OpenPose implements the bottom-up approach and by greedy parsing of the graph, it achieves real time performance. The complete architecture can be seen from Figure 2.2 and is explained in detail below.

The first part of the algorithm is to find the keypoints and joint posi-

tions from the input image. For this first task, a novel deep neural network architecture is proposed. In general, the network consists of two parts: the feature extractor and n number of stages that predict the keypoint and joint position confidence maps.

The input image I, of size (h, w), is fed into a feature extractor, in this context to a VGG-19. The output from the feature extractor is $(h/8, w/8)$ that is fed into n stages [8].

A stage is divided into two parts; branch 1 that generates the confidence maps for keypoints, and branch 2 that generates the directional fields, know as part affinity fields [8]. These two estimates give sufficient information to parse the skeleton afterwards.

Branch 1 is built from a small deep convolutional network that is upsampled to $(h/4, w/4, k)$, where k is the number of keypoints [8]. The results has as many channels as there are keypoints in the skeleton that is estimated. These confidence maps represent for each pixel the probability of a presence of a particular keypoint. For instance, one channel contains probabilities for left shoulder positions.

The output from branch 1 is fed into branch 2 with output from the feature extractor. Design is the same as in branch 1 with the difference that the network tries to predict the probable joint positions between two keypoints. In short, for each joint the branch creates a vector field between the keypoints that represents the direction of a particular limb [8]. Part affinity fields are estimated both for x- and y-direction.

As stated before, these two branches create a single stage that can be appended to the network as many time as needed. The idea behind adding more stages is that the network learns to differentiate between left and right. With a single stage, network's part affinity fields and confidence maps are able to detect body parts with the exception that left and right are the same. 3-stage network seems to be a sweet spot between accuracy and speed [8].

Loss function is constructed as a combination of a multiple L2 loss functions. Each branch of each stage gets its own loss function that is minimized. This intermediate supervision helps to address vanishing gradient problem that is present in many deep neural network architectures [20, 47]. Binary mask is also applied to the prediction to prevent penalizing true predictions [8]. This binary mask is generated from the ground truth annotations so that it masks out parts, that does not contain any limbs. The overall loss function is defined in equation 2.1. The variables are p, which is an image pixel, L is the part affinity field prediction, L^* is the ground truth part affinity field, S is the confidence map estimate, S^* is the ground truth confidence map and W is the binary mask.

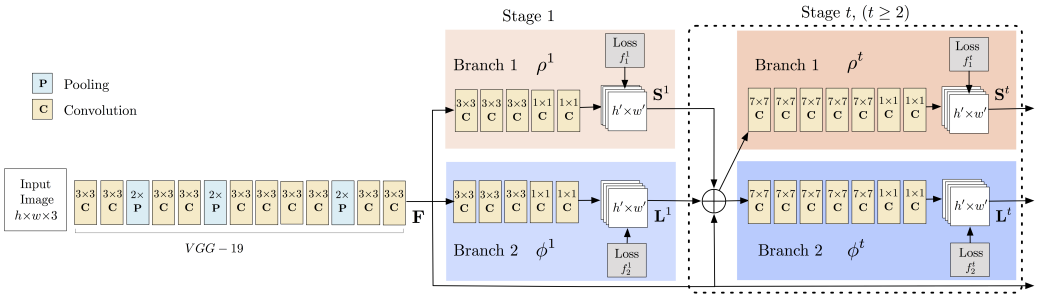


Figure 2.2: OpenPose architecture [8].

$$\mathcal{L} = \Sigma_{t=1}^{T_p} f_L^t + \Sigma_{t=T_p+1}^{T_p+T_c} f_S^t = \Sigma_{t=1}^{T_p} \Sigma_{c=1}^{C} \Sigma_p W(p) * ||L_c^{t_i}(p) - L_c^*(p)||_2^2 + \Sigma_{t=T_p+1}^{T_p+T_c} \Sigma_{j=1}^{J} \Sigma_p W(p) * ||S_j^{t_k}(p) - S_j^*(p)||_2^2 \quad (2.1)$$

An improvement to OpenPose was published in 2018 [7]. The principle is the same as previously, but the keypoint refinement has been left out. The two-branch architecture has been changed to 1-branch architecture where part affinity field estimates are first improved over multiple steps, and keypoint estimate is only attached at the end of the network [7]. The feature extractor is also changed to ResNet50. They report 45% speed and 7% accuracy improvement [7].

Keypoint estimates and part affinity fields are not enough by themselves to represent the skeleton. These estimates can be transformed into a K-partite graph, and by solving the K-dimensional matching problem one can obtain the multi-skeleton parse of K persons [8]. Vertices are the keypoints found with non-maximum suppression and edges are the possible connection candidates weighted by the part affinity field estimate. Weight value is calculated using the line integral, where the line is the the connection between the two keypoints. This matching problem is NP-hard so the matching has to be simplified to obtain results in a reasonable time [8]. This can be achieved by splitting the global matching problem into multiple bi-partial matching problems. Additionally, allowing smaller spanning trees of a skeleton than a full graph, the skeleton can be approximated quickly. The assignment problem is solved with Hungarian algorithm [8]. When the individual limb partitions has been calculated, these can be joined into larger K-partitions by connecting limbs that share body parts. The skeleton parsing can be constrained even further, by only accepting skeletons that have enough keypoints attached to it.

2.4.2 Depth based pose estimation with deep learning

Deep learning methods have been proposed to solve 2D pose estimation in depth domain [29, 44, 46]. Wang et al. developed single person pose estimation ANN using multi-task learning [46]. First part of the network predicts keypoint locations similar to OpenPose, and body part templates are found using MatchNet [46]. Tompson et al. used a combination of CNN and Markov random fields to find the skeletons [44]. Martínez-González et al. adopted the OpenPose architecture to depth images [29]. Their approach uses a custom residual network instead of the VGG-19 for feature extraction [29]. The input is normalized between $(-0.5, 0.5)$ and the depth image is preprocessed by smoothing out the discontinuities such as shadows to simplify the images [29].

Recently, most depth image based pose estimation research has been focusing heavily on extracting 3D human poses instead of 2D. Depth images are more suitable for 3D estimation than RGB as the image can be projected into 3D point cloud. Most algorithms work by transforming the depth image into 3D point cloud and use that as a input for the neural network [23, 31].

2.5 Training data simulation

Training data is a problem in pose estimation and in computer vision in general. Creating a representative dataset is an expensive and difficult task. Therefore, many researchers have chosen to generate the data with computer graphics [38, 39]. Data generation is especially tempting when working with depth images, because there is no need to model different textures and lighting conditions [26].

Shotton et al. created 15 3D human characters that they were able to vary to a few degrees [39]. They used hand collected motion capture data that they retargeted to the 3D characters to model different poses [39]. An older alternative for pose estimation used randomized joint rotations to generate the training data [38]. The benefit of using motion capture data is that the training data only contains possible human poses, but with the disadvantage that the motion capture dataset has to be huge. Generating common human poses randomly provides more flexibility, but the problem arises how to constraint the movement properly.

Martínez-González et al. adapted OpenPose to depth images by generating 230000 simulated depth images for training convolutional pose neural network [28]. They used a similar approach proposed in [39]. However, they increased the 3D base meshes to 24, created two person samples, and created

more viewpoint variability [28]. Instead of collecting own motion capture data, they used CMU motion capture dataset [32].

The obvious problem with synthetic data is that it is nearly impossible to model all imperfections that are present when using real depth cameras. There are for instance multiple sources of error that are briefly introduced in chapter 2.6. Additionally, it is impossible to represent all human variations, which leads to poor classification. However, there some interesting new approaches for dealing with this domain shift.

2.5.1 Sim2Real

Transferring computer simulated data to the real world (Sim2Real) has been studied widely especially in robotics. The goal is to minimize the reality gap between the simulation and the real world by improving the simulated data quality. System identification, domain randomization and domain adaptation are widely used techniques to tackle this issue.

In system identification, simulation is tuned to match the real world as well as possible [48]. In computer vision domain, this means generating as realistic images as possible by simulating the real world closely. This approach is expensive to compute and difficult to execute, because there are always physical effects that are not easily simulated [43].

Domain randomization takes a different approach to solve the reality gap problem. Instead of modeling the real world as closely as possible, the parameters of the simulation are varied. This means that the camera angles, textures and object shapes, to name a few, are different from sample to sample. Essentially, by heavy randomization, the simulation data distribution variance is so high that it incorporates the real world data distribution [48]. Therefore, the algorithm generalizes to work properly in the real world.

A simple approach for domain randomization is to sample simulation parameters from a uniform distribution. Tobin et al. trained an object detector using uniform distributions to simulate RGB data [43]. Their model trained on simulated data generalized to real world images, and they found out that increasing the simulation variability increases the model's accuracy [43].

Another successful example of domain randomization was introduced in [45]. They trained a car detector using a randomized non-realistic simulator that outperforms a model trained on large hand collected dataset [45]. They argue that by increasing the variance of the simulation, the neural network learns to detect the most important features, thus improving the generalization [45]. Their experiments, by fixing textures and lighting, confirm this hypothesis as the average precision decreases significantly [45].

In traditional domain randomization, the parameters are chosen from wide uniform distributions to ensure high variance. However, if we have access to real world data, we can use technique called guided domain randomization. In guided domain randomization, the real world training data is used to learn better distribution from which to sample. Thus, the ANN is less likely trained on unrealistic scenarios, which reduces the computational cost [48].

The final approach that is covered for minimizing the reality gap is domain adaptation. Domain adaptation is a process of modifying a machine learning algorithm trained with biased dataset to work properly on the target distribution. In this context, this means that we want to adapt the neural network trained on synthetic data to work as well as possible in the real world. Unsupervised deep domain adaptation was used successfully to transfer synthetic and real world data to live in the same domain in depth based pose estimation [28]. They used adversarial ANN designed for domain adaption presented in this paper [17]. The idea is to modify the pose estimation ANN so that it modifies the synthetic training images so that the domain is the same as with real world data [28].

Another well known approach for synthetic data domain adaptation was proposed in this paper [40]. Similarly to [17], they used adversarial ANN for domain adaptation, but they are using it differently. Instead of finding features that are present in both real and synthetic data, and using those for domain adaptation, they refine the synthetic data so that it looks more realistic [40].

Even though these techniques have shown great potential, they increase the training time substantially. Furthermore, depth domain adaptation is still quite poorly understood [33].

2.6 Depth imaging

The idea behind depth imaging is that instead of getting the different color channels like in traditional RGB cameras, depth cameras outputs, for each pixel, the distance to the imaging sensor. Depth cameras such as Orbbec Astra gives centimeter accuracy at close range. One of the advantages of using depth cameras instead of traditional cameras is that they allow 3D reconstruction of the scene [49]. Because the intrinsic camera parameters are known, we can apply a transformation to the obtained image coordinates

and end up with the actual 3D coordinates [49].

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_x & \gamma & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.2}$$

Equation 2.2 is the projective transformation that turns image coordinates into 3D coordinates. The variables are f, which is the focal length, γ, which is the skew factor, (x_0, y_0) defines the principal point, (x, y, z) is the camera coordinates with depth, and (X, Y, Z) are the 3D points.

Active optical depth cameras can be divided into two groups, structured light and time-of-flight cameras. Both camera types use an emitter and a receiver to measure the distance of a point in space. However, the operating principle is different for the two cameras.

Structured light cameras usually have an IR laser projector that emits a predefined pattern for instance a grid that gets deformed when hitting the objects in front of the camera [49]. These deformations are used to create a disparity map that in turn allows to estimate the z-distance to the camera plane [49]. The resolution of the camera decreases proportional to the distance.

Time-of-flight cameras calculate the distance differently. They operate by emitting a light signal and measuring for instance the phase shift when the signal reflects back to the camera. The depth value can be calculated based on that information [49].

There are multiple sources of error that affect the performance for both optical depth sensor types [37]. Ambient background light, multi-device interference and temperature drift are common error sources for both types to name a few [37].

A noticeable problem with structured light cameras is the quantization of the disparity map [37]. This means that the depth values get projected into discrete planes. The depth values become discontinuous and the distance between these quantized planes increases proportional to the depth value.

Chapter 3

Methods

This chapter explains the implementation choices and modifications to the OpenPose architecture to make it more suitable for depth based pose estimation. Data simulation process is explained in detail as well. The whole pipeline is summarized in Figure 3.1.

3.1 Architecture

The deep learning algorithm used in this thesis is the one presented in the original OpenPose paper [8]. An existing implementation is used as a starting point provided by the TensorLayer team [13]. The following paragraphs explain the modifications that are implemented to make it more suitable for depth based pose estimation.

The architecture is modified so that different feature extractors can be used instead of the default VGG-19. MobileNet was chosen as an alternative, because it is designed for embedded and mobile applications [24]. As discussed in chapter 2, keypoint and part affinity field estimates are refined by increasing the number of consecutive stages. The trade-off is that we get better accuracy while increasing the computational cost of the training process. In this implementation these stages have been reduced from 6 stages to 5 stages. Stages are reduced to speed up the training process. The improvement after increasing the stages above 3 is limited, but with simpler feature extractor, such as MobileNet, the computation cost should stay reasonable [8].

Hyperparameters for the neural network are kept the same as in the reference implementation [10]. Adam optimizer is used with exponential decaying learning rate. Learning rate starts at $5 * 10^{-4}$, decay factor is 0.33 and the learning rate is decayed every epoch.

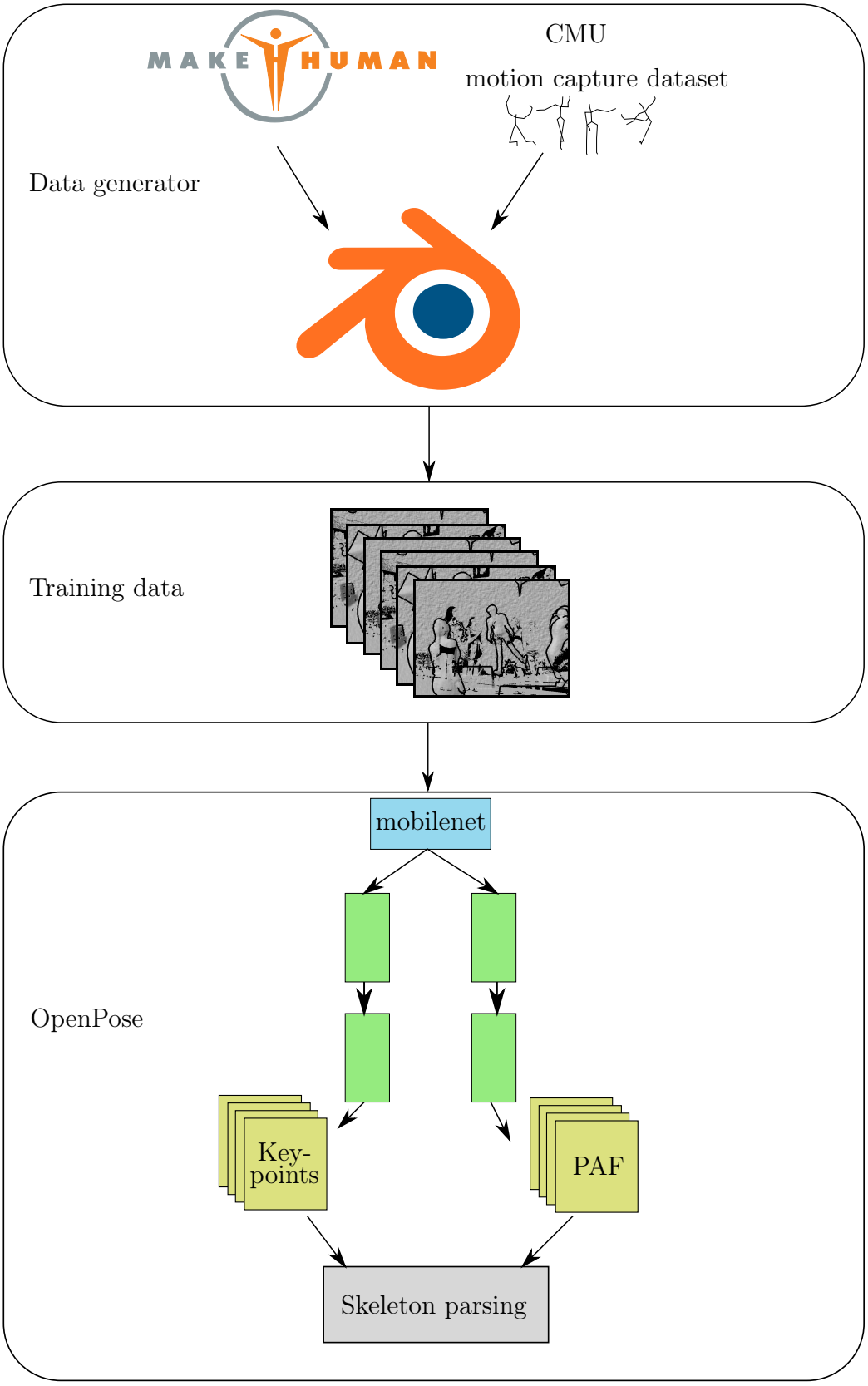
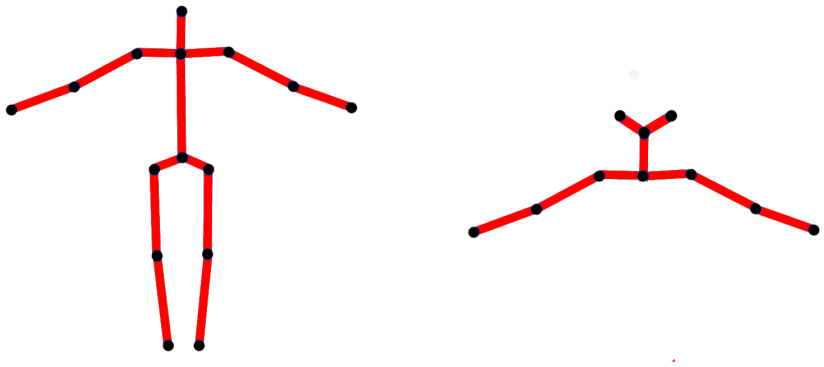


Figure 3.1: Pipeline



(a) Full skeleton (b) Skeleton for truck use case

Figure 3.2: Skeleton models

The original skeleton consisted of 18 keypoints and 19 part affinity fields [8]. However, it can be argued that detecting eyes and ears from a depth image is harder than from an RGB image, because depth images lack texture. Resolution of the Orbbec Astra camera is quite poor after 2 meters, so beyond that range, it becomes increasingly more difficult to detect fine details. Therefore, it is justified to simplify the skeleton model to address this issue by removing these 4 keypoints. Thus, we end up with two different simplified skeleton models, the other one for the meeting room use case, and the other for truck driver detection. There are now 15 keypoints for the first use case and 10 for the latter. For truck driver detection use case, eyes are still visible, because the range stays below 2 meters. Skeletons can be seen in Figure 3.2.

3.2 Post processing

As discussed in chapter 2, part affinity fields and keypoints estimates are transformed into a skeleton by greedy parsing of the graph. The algorithm is controlled by 5 different parameters. Keypoints are extracted from the heatmap by calculating non-maximum suppression. Sensitivity of accepting a keypoint is controlled by a threshold parameter. Similarly, part affinity fields are thresholded, with the idea that weak vector fields should be discarded.

Additionally, weak skeletons are removed if the part and limb counts are below a given threshold or the total score that is calculated as a sum of part scores is also below a threshold. The choice of these values is quite arbitrary and is tuned by hand.

These parameters are the hyperparameters of the model. Therefore, these can be optimized by minimizing the estimation error against the validation dataset.

3.3 Data simulation

One of the major problems in machine learning is collecting a representative real world datasets with good quality annotations. Quite often data samples have to be manually labeled, which makes it a tedious and time consuming task. This is especially problematic with deep learning algorithms as they require huge amounts of data to be properly optimized.

A good solution to this problem is to use simulation tools that can mimic real world situations for generating the training data. Simulation is especially appealing for depth cameras, because it is simple to generate life-like images. Depth data is invariant to different lighting conditions and objects are textureless, which simplifies the simulation process. Ground truth data will also be extremely accurate as the skeleton inside the model is static in a sense, so the projected coordinates do not have variation that might occur when a human is labeling the data.

In this thesis, domain randomization technique is used to decrease the reality gap. As discussed previously, the idea is to create a data simulator with high randomization so that a sample from a real world is just one variation in the simulation [48].

The data was simulated using an open source 3D modeling software called Blender [12]. 500 different human characters, with different body types and clothing were randomly generated using MakeHuman tool [4]. Using multiple different models provides variance to the data that allows the model to generalize better. Different poses were generated by loading a random animation from the CMU Motion Capture dataset and choosing a random frame from that animation. CMU dataset contains motion capture data from different everyday scenarios that should represent common human poses [32]. The dataset consists mostly of people walking and sitting while doing some simple task. Additionally, each joint is rotated randomly between $-3, 3$ degrees after loading the poses.

The problem with CMU dataset is that the variability is quite poor. Therefore, random poses with constraints were added to the training dataset.

Furthermore, the simulation pipeline generates images with multiple persons as well. One of the biggest challenges in multi-person pose estimation is person occlusion. By adding a lot of these examples in the training data should help the model to generalize better.

Poses for the truck driver use case are simulated differently. Head and arm joints are rotated randomly within constraints, instead of loading the CMU animations. The variability of upper body poses is quite limited in the dataset, so it is reasonable to rotate them randomly to get better variance in the training set.

To obtain variance to scale and viewpoint, the camera is translated and rotated randomly to different orientations. The movement is constrained to represent the actual camera positions in the two use cases. In the meeting room use case, the camera is close to the ceiling and looking downwards. In the truck driver use case, the camera is in front of the person, withing 2 meters.

One possible difficulty for depth based pose estimation is the lack of detail for instance in human face, because the camera loses depth resolution rapidly. Therefore, it is more difficult to distinguish certain shapes for instance human head from a sphere. Thus, the simulation should provide some ways to add random objects to the scene so that the network learns the most important features. This problem is solved by adding a simple particle system that spawns different primitive objects such as cubes and spheres to the scene. The variability is increased even further by adding random displacement modifiers to the objects with varying noise textures. The idea of flying distractors was proposed in [45]. With this modification, the dataset also contains samples where the person is only party visible. Occlusion is common in real world scenarios, so it is useful that the training data reflects this as well. Randomization was also added to the floor and back plane by applying a random displacement modifier that creates random shapes to those planes.

Depth cameras are not perfect, thus they are not capable of measuring the depth value in every situation as discussed in chapter 2.6. One main reason for missing depth values is, that objects close to the camera cast shadows on the objects that are more distant. This happens, because the emitter in the depth camera is located besides the receiver. This aspect is easily modeled in the 3D simulator by moving a point light source around the camera and capturing the casted shadows.

Another approach for background augmentation would be to add a 3D human on top of a real world image [29]. There are 3D datasets available, but they use different sensor, thus the accuracy would most certainly worsen for the sensors used in this thesis.

3.3.1 Error estimation

The depth images coming from the simulation are extremely smooth, as the models are not accurate enough to model for instance wrinkles in clothes. A top of that, current depth cameras such as Orbbec Astra, that is used in the meeting room use case, loses accuracy quite quickly as the distance increases. On the other hand, Azure Kinect, that is used in the truck driver use case, has a much better quality at closer range and the error is much less. For these reasons, a comprehensive error model was developed based on Orbbec Astra sensor. The error model used for Azure Kinect is simpler. The error model for Orbbec Astra is quite simple and follows roughly the findings presented in this paper [18]. Even though our camera is different, our experimentation shows similar behavior. Another approach to deal with the noise is by filling out the noise and shadows by inpainting process [29]. Furthermore, the noise in clothing could be estimated by adding random noise to 20% of pixels [29].

The developed error model works as follows. First, Gaussian noise is added to the depth values. Variance of the Gaussian distribution increases quadratically by this formula $1 * 10^{-6} z^2$, where z is the depth value. The formula is rather arbitrary, but has a solid intuition behind it. It was reported in the paper that the systematic error of a structured light depth camera increases quadratically [18]. The paper provides a formulation, however the error increases too rapidly for the camera used in this thesis. With the camera used in this thesis, the error is around ± 10 cm in 7 meters which is more consistent with own experiments with the camera.

There is also a random error component for the depth. In this case, more error is added again with Gaussian distribution $\mathcal{N}(0, 2)$. This error term is irrelevant for longer distances, but is useful for shorter distances to roughen the surfaces. In the real world, surfaces are hardly never completely smooth, so it is useful to reflect this property as well.

Finally, it was observed that the resolution of the depth camera decreases again quadratically. This was observed by taking an image of a wall perpendicular to the image plane. With this setup, by projecting the image into 3D point cloud, it is simple to observe and measure how the resolution decreases. For Orbbec Astra sensor, points are located on discrete planes. This behavior can be seen in Figure 3.3. The distance between these planes keeps increasing the further away the point is from the camera. The distance between these planes increases by this formula $3.997 * 10^{-6} z^2 - 1.656 * 10^{-3} z + 10.06$ where z is the depth value. This estimate was created by manually measuring the distance between these planes for different distances in CloudCompare software [19]. Then, a simple quadratic function was fitted to these observations by minimizing the least squares. Observations and the fit is presented in

Figure 3.4.

By combining these previously presented error estimates we end up with the error model. The simulated images now represent real world depth images substantially better. See algorithm 1 for more details. Examples of synthetic images can be seen from Figure 3.5 and 3.6.

The error model for Azure Kinect uses the same quadratically increasing Gaussian noise and random error component as presented above. On the other hand, the discretization of depth values is left out.

Algorithm 1 Apply error model for depth value z

$i \leftarrow 0$
$plane \leftarrow [0..8000]$
while $i < 8000$ **do**
    Find the plane for depth i by applying $q = 3.997 * 10^{-6} * i^2 - 1.656 * 10^{-3} * i + 10.06$
    increase i until i is larger than $i + q$ while populating $plane$

function ERROR(z)
    if $z < 400$ **then**
        return 0
    $\sigma_{systematic} \leftarrow 1 * 10^{-6} z^2$
    $\epsilon_{systematic} \leftarrow \mathcal{N}(0, \sigma_{systematic})$
    $\epsilon_{random} \leftarrow \mathcal{N}(0, 2)$
    $\epsilon_{combined} \leftarrow \epsilon_{systematic} + \epsilon_{random}$
    $z_{quantized} \leftarrow plane[\epsilon_{combined}]$
    return $z_{quantized}$

3.4 Augmentation

Data augmentation is a process that tries to increase the usefulness of each training sample by applying different transformations to the data. In the context of computer vision this means that training images are cropped, scaled and rotated randomly within certain constraints. These transformations help the network to be more robust to scale, rotation and translation.

3.4.1 2D data augmentation

In this thesis, the transformations are constrained in a similar fashion as in the original OpenPose paper [8]. The rotation is from $-30°$ to $30°$ and zoom is from 0.5 to 1.0. For computational efficiency, these transformations are

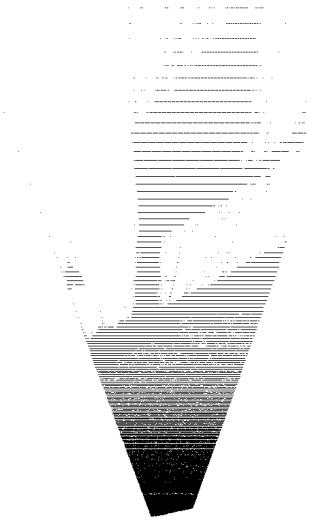


Figure 3.3: Side view of the point cloud generated from the wall image. The distance from the camera increases up. Distance between the lines increases quadratically.

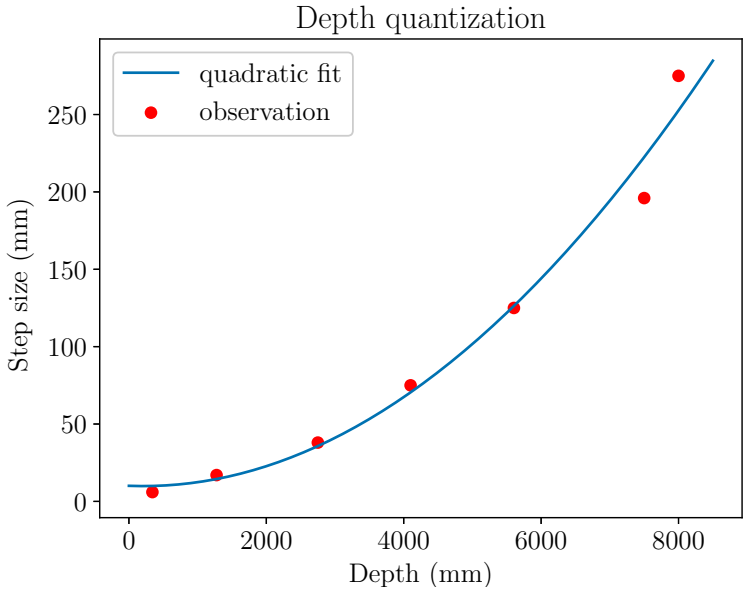


Figure 3.4: Estimation of Orbbec Astra S depth camera's depth quantization error. X-axis is the distance from the camera and Y-axis is the distance between the quantized planes. The observations are based on the point cloud in Figure 3.3.

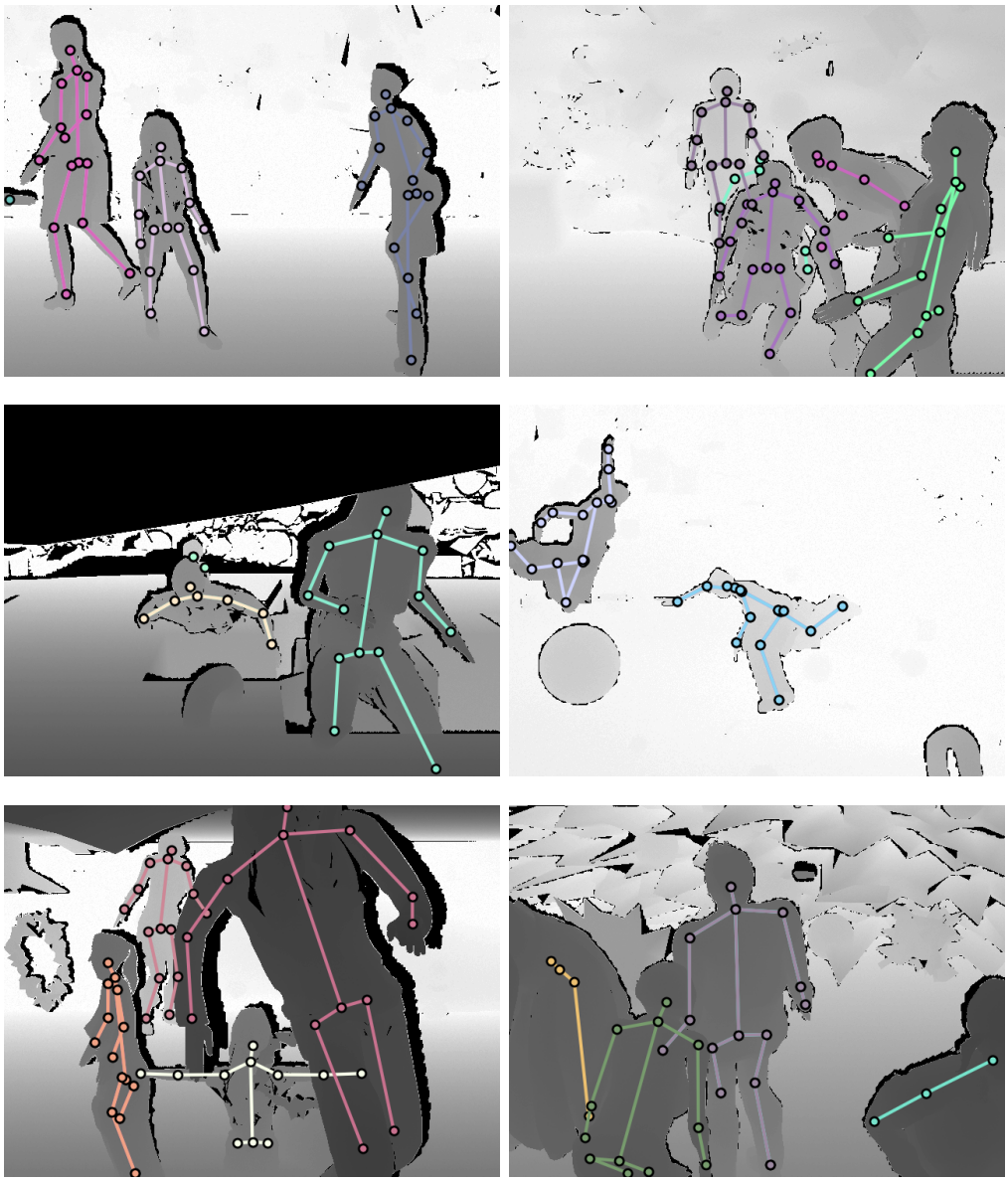


Figure 3.5: Example of synthetic data for the meeting room use case

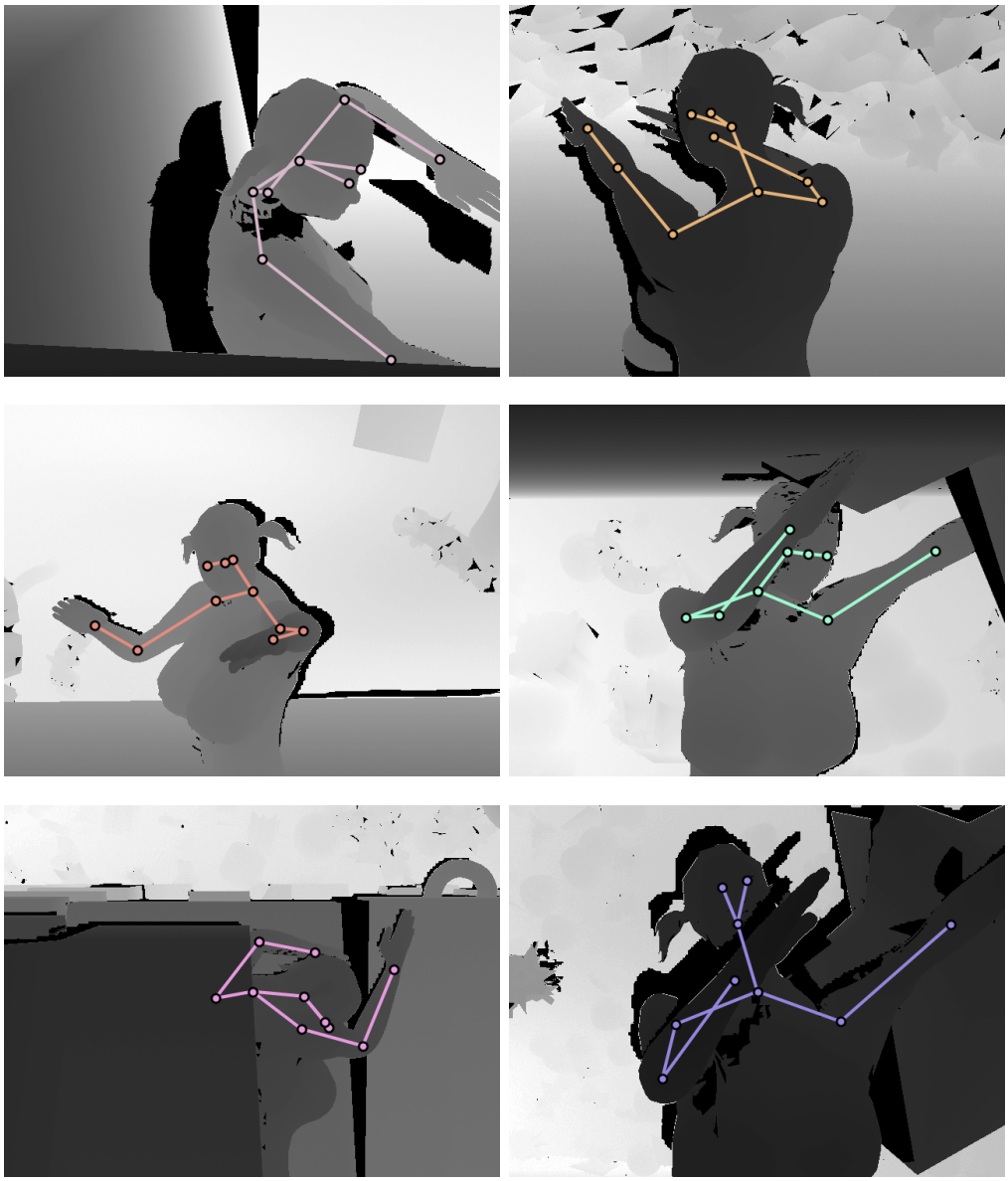


Figure 3.6: Synthetic data for the truck driver use case.

(a) no mask (b) mask

Figure 3.7: Effect of noise mask

combined into a single affine transformation [13]. Brightness and contrast variations are removed, because the depth camera is relatively invariant to these changes.

3.4.2 Noise mask

Depth data is not perfect. The data contains many areas where the sensor is not able to calculate the correct distance and reports zero distance instead. Certain materials, such as black jeans are invisible to the Orbbec Astra S camera. Therefore, the dataset is augmented with a noise mask that turns some depth values to zero. Noise masks are generated by loading a random image as grayscale, and finding the percentile from range $U(0, 25)$. Then, the image is thresholded based on that value and turned into a binary mask. The final result is a simulated depth image that has $0 - 25\%$ of pixels turned into zero. The random images are taken from the COCO dataset [27]. Example of a random noise mask can be seen from Figure 3.7.

3.5 Preprocessing

Neural networks have millions of parameters that should be optimized. Often it is useful to start the training process by loading pretrained weights that are trained on a huge dataset and finetune the network using a smaller custom dataset. This approach gives a better starting point for the network so the training time should decrease. It can be also argued that the features that the first layers of the CNN learns are similar for most of the networks, so it is unnecessary to train these multiple times. Another consideration is that when

the depth of the network starts increasing, it becomes harder for the gradient to flow to the first layers. This means that it is also harder to train these initial layers properly without custom training procedures such as freezing parts of the network and training only subset at a time. Nevertheless, most modern CNN architectures have structures such as identity connections to deal with this problem, but with networks such as the OpenPose, one can still encounter issues due to the depth of the network.

For these reasons, the feature extractor part of the network is initialized with pretrained weights. The loaded weights are trained with ImageNet dataset for MobileNet version 1 and VGG-19. Since the ImageNet dataset is a RGB dataset, there are 3 input channels for the pretrained feature extractors. Hence, the depth image has to be preprocessed to have 3 channels.

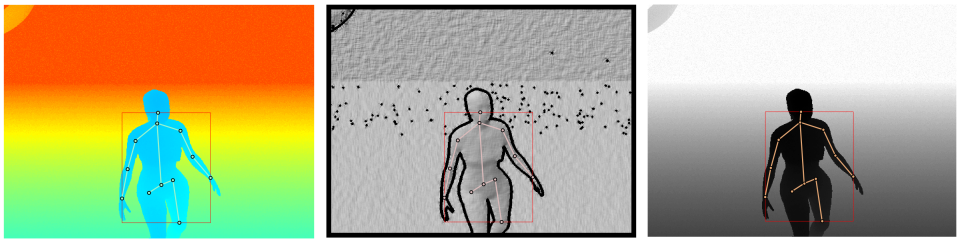
The most obvious approach is to normalize the data and replicate it to 3 channels. Another way is to use colormaps to encode the depth information into 3 channels. Jet colormap was chosen for experimentation as it has been used previously with success in image classification [14]. Two different coloring ranges are tested $(0, 2000)$ and $(0, 10000)$. As discussed previously the maximum range of the used sensor is 8000 mm. With the longer range we are able to cover the whole distance, but we lose some resolution in the process. On the other hand, the shorter range gives better resolution, which is useful in the truck driver use case.

3.5.1 Normal estimation

None of these previous coloring methods utilize the available depth information. As noted before, point cloud construction is a trivial task that allows new possibilities for the data augmentation. Similarly, from the point cloud it is possible to estimate the normal maps of the image. Normal maps transforms the data samples into scale invariant estimates that should help the neural network to learn as the input data is simpler in a sense. However, as discussed before, the resolution of the camera decreases rapidly when the distance increases. Therefore, the normals estimated for long distances are poor. Large step size between the planes makes it nearly impossible to estimate them properly. Nevertheless, normal estimation works well for distances below 4 meters so this is a valid approach for use cases where environment is constrained such as in the meeting room setting.

Furthermore, normal maps have been used successfully for depth coloring multiple times [2, 6]. They have been shown to outperform jet coloring in image classification.

In a simplest sense, normals can be estimating by finding the line perpendicular to a tangent plane of a point in the object's surface. The plane can



(a) jet coloring (b) normal coloring (c) no coloring

Figure 3.8: Different preprocessing techniques

be found by calculating the eigenvalues of the covariance matrix estimated from the surrounding points. These eigenvalues and eigenvectors define the plane, which cross product is the normal.

Normals are calculated using PCL library's normal estimation using integral images function [36]. The implementation offers three different options to estimate the normal of a point. In this case, the *AVERAGE_3D_GRADIENT* option was used. This approach calculates smoothed 3D gradients to construct the tangent plane.

The computational cost of this preprocessing step is minimal, as the code is implemented in C++ and interfaced efficiently with Python. The only drawback is that the normal estimate has to be replicated to 3 channels for pretrained weights, which makes the network deeper.

As a side note, one interesting approach would be to use machine learning to learn a mapping from a 1 channel depth image into 3 channel color image. One such example uses deep ANN to generate colormaps that preserves interesting depth information better than colormap jet and normal map, thus giving better object classification accuracies [9].

Chapter 4

Evaluation

4.1 Test data

Hand annotated test dataset was created for evaluating the performance in the real world. Dataset can be seen from Figure 4.2. The dataset consists of 115 depth images with different backgrounds and poses. The dataset contains both single and multi-person examples. Camera is always positioned high to simulate the meeting room use case. The data is collected using Orbbec Astra structured light camera.

Similarly, a small dataset was collected for the truck driver use case. Images can be seen from Figure 4.3. The dataset consists of 40 samples. Dataset is small because the possible poses are extremely limited. Hands are usually on the steering wheel hardly never moving so the variation can be presented with a few images. The dataset is collected using Azure Kinect time-of-flight camera.

4.2 Setup

The experiments are run with different simulation setups, image coloring techniques and feature extractors to get a better understanding of their importance in the meeting room use case. The experiments run on the truck driver use case are restricted based on the findings in the meeting room use case to save computational time.

Meeting room use case training data consists of 250000 images and the truck driver use case has 220000 training samples. Training data is generated before training the networks.

Experiments are run with batch size of 5 and saved every 10000 steps. Every network is trained at least 120000 steps to ensure sufficient conver-

Figure 4.1: Hand annotated validation dataset for meeting room use case.

Figure 4.2: Hand annotated test dataset for meeting room use case.

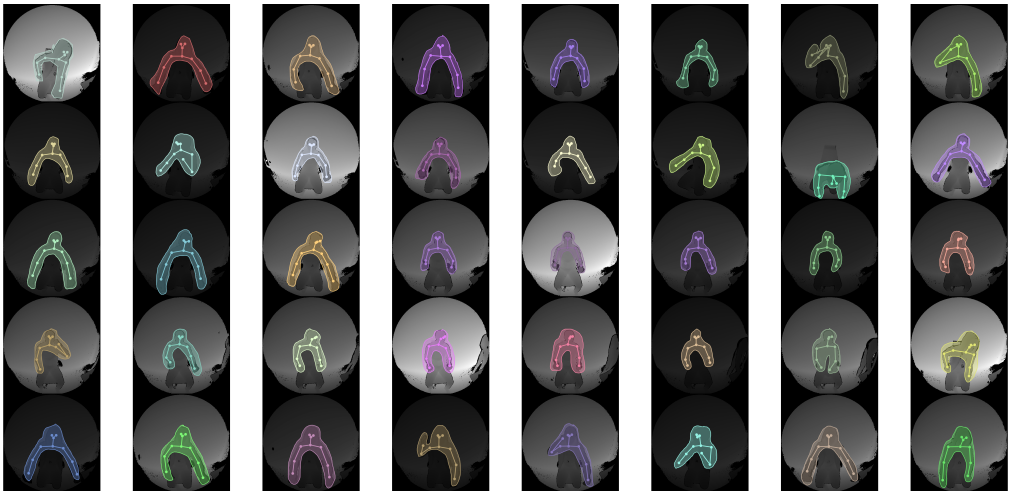


Figure 4.3: Hand annotated test dataset for truck driver use case.

Parameter	Distribution
PAF length	$\mathcal{U}(1, 14)$
Non-maximum suppression threshold	$U(0, 0.5)$
PAF threshold	$U(0, 0.5)$
Limb(/PAF) count	$\mathcal{U}(1, 14)$
Part(/keypoint) count	$\mathcal{U}(1, 15)$
Part(/keypoint) score	$U(0, 1)$

Table 4.1: Search space for hyperparameter optimization.

gence. Computer specifications are Ubuntu 18.04.2 64 bit, Intel Core i9-9900K, Nvidia Titan RTX and 32 GB RAM.

As discussed previously, the depth camera chosen for meeting room use case is the Orbbec Astra. It operates using structured light principle, has a range up to 8 meters and outputs 640x480 depth images at 30 fps [1]. Orbbec Astra is more suitable for mobile applications than perhaps the most common depth sensor Microsoft Kinect due to the lower power consumption and power over USB.

Azure Kinect, that is chosen for the truck driver use case is more modern depth camera that operates using time-of-flight principle [30]. It has larger field of view and better resolution at close range, which makes it more suitable for this use case.

Hyperopt package is used for finding the hyperparameters of the model [5]. The optimization is run on the best performing model based on the validation accuracy. A search space is specified in Table 4.1. Optimization algorithm is the default tree-structured Parzen estimator. These results are then compared against the reference values from the original implementation [10]. Validation dataset consisting of 218 hand annotated pictures and is only performed for the meeting room use case. Images can be seen from Figure 4.1.

4.3 Metrics

Keypoint accuracies are measured with two metrics. The first one calculates the object keypoint similarity (OKS) metric that is defined in equation 4.1. The parameters are $\hat{\theta}^p$, that is the estimated keypoint location, θ^p, that is the ground truth annotation, k_i, that is user defined standard deviation for the Gaussian distribution for keypoint i, s, that is the scale normalization constant (ground truth annotation segmentation area) and v_i is an indicator $s \in \{0, 1\}$ representing if a keypoint is labeled [35]. OKS value is also thresholded with a value between $(0, 1)$.

$$\begin{cases} ks(\theta_i^p, \hat{\theta}_i^p) = e^{-\frac{||\hat{\theta}_i^p - \theta^p||_2^2}{2s^2k_i^2}} \\ OKS(\hat{\theta}^p, \theta^p) = \dfrac{\Sigma_i ks(\hat{\theta}_i^p, \theta_i^p)\delta(v_i > 0)}{\Sigma_i \delta(v_i > 0)} \end{cases} \tag{4.1}$$

OKS only gives a performance evaluation for a single image so traditionally in keypoint estimation, mean average precision (mAP) is calculated. mAP is defined in Equation 4.2, where Q is the list of keypoint scores, here OKS scores, and $AveP$ is the average precision. In this thesis the variance k is 0.5 for each keypoint and OKS threshold is 0.5.

$$MAP = \frac{\Sigma_{q=1}^{Q} AveP(q)}{Q} \tag{4.2}$$

Performance of the model is also evaluated with percentage of correct keypoints (PCKh) metric [3]. Body part is matched if the end keypoints of that particular limb are within 50% of head segment length [3]. The idea is to make the metric independent from articulation [3]. In this thesis the keypoint normalization is changed to neck-hip length to allow larger error for the meeting room case. The normalization distance is neck to nose in truck driver use case.

Chapter 5

Results

5.1 Meeting room use case

5.1.1 Hyperparameter optimization

The network chosen for hyperparameter optimization is based on the validation accuracies presented in Figure 5.1. It is clear that VGG-19 outperforms MobileNet, and normal coloring seems to be the best choice. VGG-19 works in real time, which makes using MobileNet unjustified. Showing all keypoints in the simulated data if half of the keypoints are visible is better, than always hiding them if they are occluded or always showing every keypoint. Additionally, the simulation data contains a lot of fully occluded examples where no keypoints are visible, which makes it unjustified to show every keypoint in all situations. Hyperparameter optimization is done for the ANN that is using VGG-19, normal coloring, and simulation setup where keypoints are shown if 50% are visible. Additionally, shadows are modeled in the simulation. The result from the hyperparameter optimization can be seen from Figure 5.2 and the found optimal values from Figure 5.1.

5.1.2 Test set results

Results against the test set can be seen from Table 5.2. Results are similar to those against the validation data. MobileNet is significantly worse than VGG-19, and normal coloring seems to work the best. Shadow modeling seems to improve the performance slightly. Additionally, hyperparameter optimization seems to give a minor improvement as well.

PCKh results for individual keypoints for the best performing network can be seen from Figure 5.3. It can be seen that the hardest keypoints are

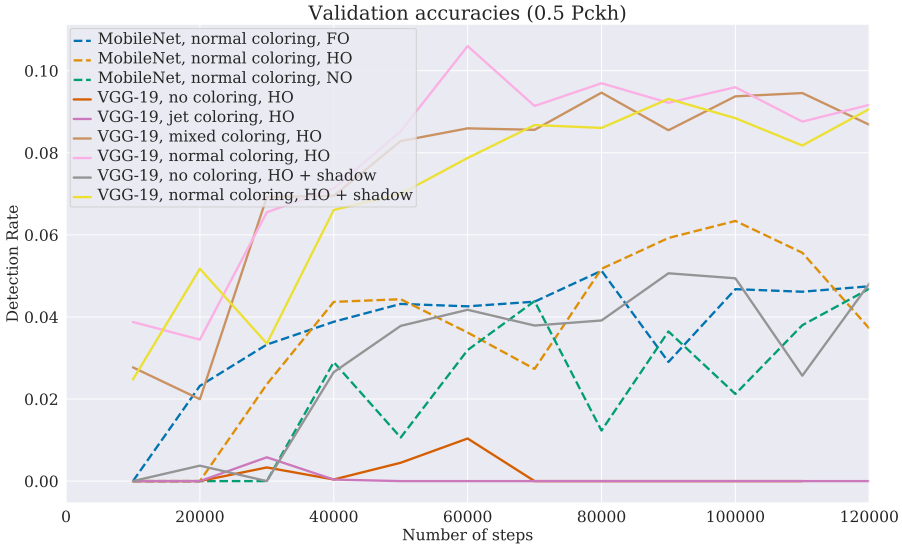


Figure 5.1: MobileNet and VGG-19 were used as feature extractors. Four different coloring techniques were tested, where no coloring means only normalizing the image between $[0, 1]$ and mixed meaning that input had one channel as normal map and another two as normalized depth channels. Finally, the last symbols represent different simulation setups, where FO means full occlusion (always hide keypoints if they are not visible to the camera), HO means half occlusion (if 50% of keypoints are visible show them all) and NO means no occlusion (always show all keypoints). Additionally, shadow means adding the randomized shadow areas to the image.

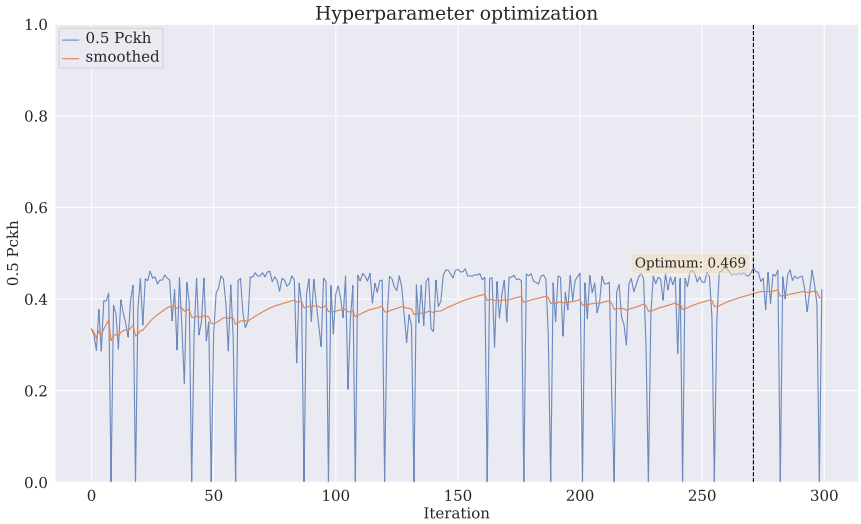


Figure 5.2: Hyperparameter optimization using 0.5 Pckh as an evaluation metric.

Parameter	Optimal	OpenPose default
PAF length	10	10
Non-maximum suppression threshold	0.017266	0.1
PAF threshold	0.000388	0.1
Limb(/PAF) count	7	5
Part(/keypoint) count	5	4
Part(/keypoint) score	0.13835	0.6

Table 5.1: Optimal hyperparameters based on hyperparameter tuning. OpenPose default values presented for comparison.

ANN			AP OKS	0.5PCKh
Coloring	**Simulation**	**Feature extractor**		
Normal	full occ.	MobileNet	0.531	0.077
Normal	half occ.	MobileNet	0.209	0.045
Normal	no occ.	MobileNet	0.394	0.065
Depth	half occ.	VGG-19	0.050	0.010
Jet	half occ.	VGG-19	0.008	0.003
Normal	half occ.	VGG-19	0.722	0.177
Mixed	half occ.	VGG-19	0.717	0.163
Depth	half occ.+shadow	VGG-19	0.538	0.085
Normal	half occ.+shadow	VGG-19	0.771	0.185
Normal	**half occ.+shadow**	**VGG-19**	**0.829**	**0.199**

Table 5.2: Test set accuracies for different neural networks. Bolded row is after hyperparameter optimization against the validation dataset.

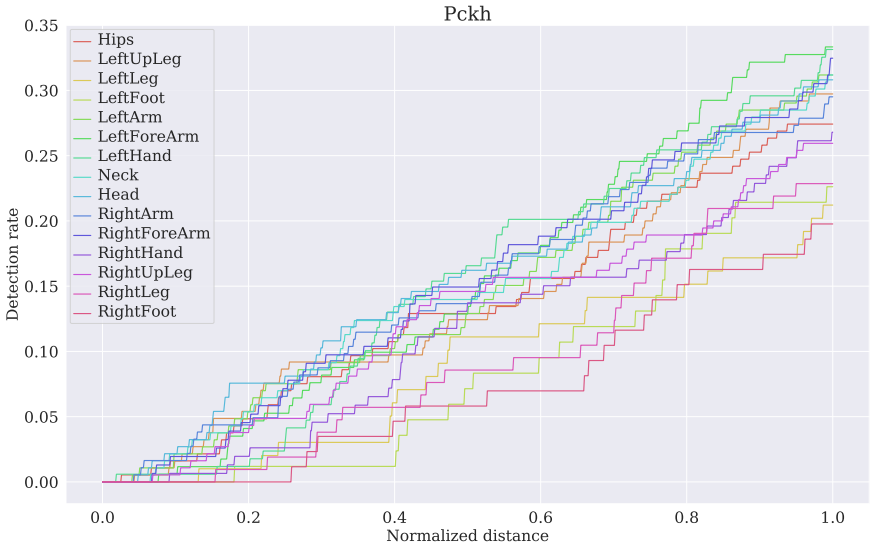


Figure 5.3: Best performing ANN for meeting room use case.

the ones, which have the least connections. These are the outer limbs such as feet.

Reported results in a similar depth image OpenPose implementation show better results [28, 29]. However, numerical comparison is difficult in this case due to different evaluation metrics, different validation sets and slightly different skeletons.

5.2 Truck driver use case

Early experiments with Orbbec Astra structured light camera showed poor results. Therefore, the experiments were conducted using Azure Kinect time-of-flight camera. Results can be seen from Table 5.3. Network without any depth coloring works the best, compared to the meeting room use case where normal coloring gave a slight improvement.

Individual keypoint accuracies for the best performing network can be seen from Figure 5.4. Network detects eyes, nose and neck with the greatest accuracy. Left arm is detected slightly better than the right arm. Pckh scores are better in this use case than in the meeting room use case. This is expected, because the use case is simpler.

ANN		AP OKS	0.5PCKh
Coloring	**Feature extractor**		
Depth	VGG-19	0.792	0.456
Jet	VGG-19	0.284	0.162
Normal	VGG-19	0.308	0.325

Table 5.3: Test set accuracies for truck driver use case.

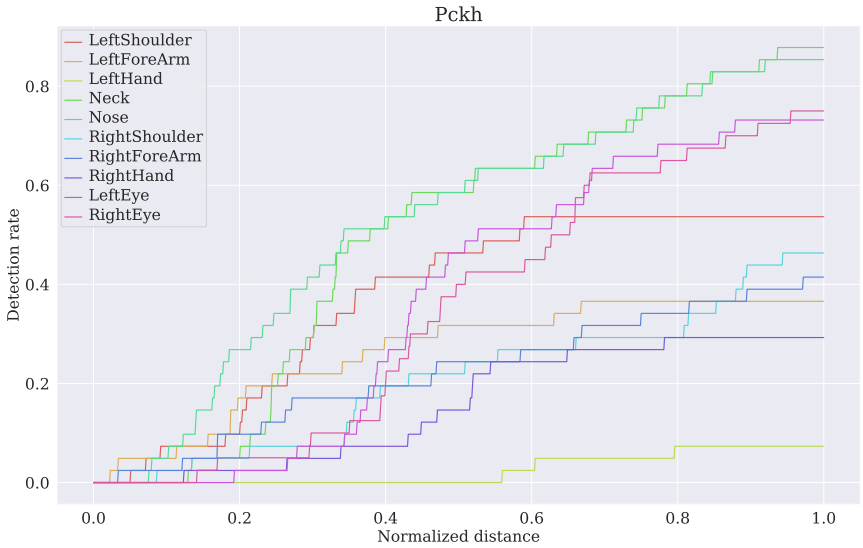


Figure 5.4: Best performing ANN for truck driver use case.

Chapter 6

Discussion

In general, meeting room use case works well in the real world. Occlusion handling is surprisingly robust in many different scenarios. Network can handle people sitting around a table, even though the training data does not contain any explicit examples. The network can predict the whole skeleton through occlusion as well.

The effects of coloring are quite clear. Normal coloring worked the best. Most likely it simplifies the problem by transforming the depth image into depth-invariant normal map. It also emphasizes the edges, which are more representative features.

VGG-19 outperforms MobileNet substantially. Even though MobileNet is expected to perform worse, because it is much smaller, the results show much larger difference in pose estimation. The benefits of using MobileNet are slim, because the inference time is dominated by the skeleton parsing. Forward pass in MobileNet can be done in 100 fps, while forward pass is 60 fps in VGG-19. On the other hand, when we do the skeleton parsing as well, MobileNet operates at 50 fps and VGG-19 at 40 fps.

One possible reason for poor performance with MobileNet might be due to the domain randomization utilized in the data generation. We are essentially training the network to work in unrealistic situations, which in turn requires more learning capacity from the model. OpenPose architecture using VGG-19 is nearly 30% larger than OpenPose with MobileNet, which explains the results.

The importance of showing keypoints if occlusion is present is difficult to determine based on the test set results. Test set and validation results are conflicting, which is most likely due to the small dataset sizes. However, manual experimentation shows that the simulation data should have occluded keypoints visible, because the network obviously learns to predict hidden keypoints based on the others. This is not the case, if keypoints are always

invisible when occlusion is present. In future work, the threshold value for showing all keypoints should be experimented more carefully, because the current choice for 50% is arbitrary.

Outer limbs are classified worse than the others that have more connections. This might be due to the OpenPose's architecture. The network tries to model the relative locations of the keypoints by predicting the part affinity fields. However, this approach favors keypoints that are connected to multiple points. In other words, keypoints such as neck and hip are connected to multiple different keypoints, which means that there are multiple PAFs that predict the position of that keypoint. Thus, keypoints such as ankles and wrists, rely only on a single PAF for prediction. A natural fix would be to add additional connections between keypoints to improve the network's ability to model spatial dependency. For instance the OpenPose implementation has a connection between shoulders and ears [10]. This relationship stays relatively fixed, which means that the ANN has a strong "guess" where the other keypoint is, if the other is occluded. However, choosing these additional connections is not a trivial task. Adding additional connections means that the ANN becomes deeper and the optimization problem becomes harder, which in turn might require more data. Moreover, the connected keypoints should be strongly correlated spatially such as an ear and a shoulder. One could suggest for instance to join a shoulder and a wrist, because the elbow joint restricts the movement substantially.

The mixing problem in multi-person occlusion setting is a common problem in pose estimation. Even though OpenPose's architecture refines the keypoint and PAF estimates, it still struggles in some situations. In this case, the problem is a bit over represented, considering we are using the same network design. This is caused by training data and domain shift. Even though, training data contains multi-person examples similarly to [28], current domain randomization is obviously not high enough. Furthermore, it is impossible to simulate perfectly the real world, which results in a different learned domain for the network. In future, this domain shift could be dealt using adversarial training approach that has been shown to work for simulated depth data [28].

The synthetic data generation could easily be developed even further. One such improvement could be by simulating different material properties. Certain clothes, such as black jeans are invisible to Orbbec Astra, because they absorb the emitted grid.

Manual experimentation showed poor performance at distances longer than 4 meters from the camera. Most likely cause for this is improper error modeling at longer distances. This could be solved by adding more randomization to the error modeling, especially to quantized depth steps. They are

currently fixed.

Truck driver use case works as it was planned. The only noticeable problem is poorer performance when predicting right side of the body. The network confuses left and right, which is an indication that the stage refinement in OpenPose architecture does not work well enough in this context. This could be solved by increasing the number of stages.

Early experiments with Orbbec Astra showed poor results in truck driver use case. However, as can be seen from the Table 5.3, Azure Kinect worked moderately well. The failure with Orbbec Astra is an example of poor domain randomization at close range. The error model is obviously not able to capture the real world behavior properly when the distance is below 2 meters. On the other hand, Azure Kinect works, because the quality of the depth sensor is much better than the one in Orbbec Astra. Depth images from Azure Kinect resemble simulated data better, which makes it less dependent on properly randomized error modeling.

Chapter 7

Conclusions

In this thesis, we adopted a real time multi-person deep learning algorithm to detect human poses from a single depth image. The network was trained on computer simulated data that utilized domain randomization techniques to bridge the gap between simulation and the real world. Trained models work in occluded multi-person scenarios.

It is clear, that a model trained on synthetic data can generalize properly to the real world scenarios without additional post processing techniques. The generated synthetic dataset pipeline should produce images that are so randomized that they are difficult to annotate even for a human. It seemed that increasing the randomization increased the model's accuracy as well. The learning capacity of the neural network has a significant impact on the performance as well when using synthetic data.

Future work should focus on improving the simulation pipeline. Simulation should be randomized even further to force the network to focus only on the most important features. Furthermore, guided domain randomization could help scaling to smaller neural network architectures.

Bibliography

[1] Orbbec astra s. https://orbbec3d.com/product-astra-pro/. Accessed: 20.6.2019.

[2] Aakerberg, A., Nasrollahi, K., Rasmussen, C. B., and Moeslund, T. B. Depth value pre-processing for accurate transfer learning based rgb-d object recognition. In *IJCCI* (2017).

[3] Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE meeting on Computer Vision and Pattern Recognition (CVPR)* (June 2014).

[4] Apache Software Foundation. Makehuman. http://www.makehumancommunity.org/. Accessed: 15.10.2019.

[5] Bergstra, J., Yamins, D., and Cox, D. D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International meeting on International meeting on Machine Learning - Volume 28* (2013), ICML'13, JMLR.org, pp. I–115–I–123.

[6] Bo, L., Ren, X., and Fox, D. *Unsupervised Feature Learning for RGB-D Based Object Recognition*. Springer International Publishing, Heidelberg, 2013, pp. 387–402.

[7] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008* (2018).

[8] Cao, Z., Simon, T., Wei, S., and Sheikh, Y. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR abs/1611.08050* (2016).

[9] Carlucci, F. M., Russo, P., and Caputo, B. Deco: Deep depth colorization. *IEEE Robotics and Automation Letters 3*, 3 (2018), 2386–2393.

[10] Charles, P. Openpose-plus. `https://github.com/tensorlayer/openpose-plus`, 2018.

[11] Chen, L., Wei, H., and Ferryman, J. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters 34*, 15 (2013), 1995 – 2006. Smart Approaches for Human Action Recognition.

[12] Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[13] Dong, H., Supratak, A., Mai, L., Liu, F., Oehmichen, A., Yu, S., and Guo, Y. TensorLayer: A Versatile Library for Efficient Deep Learning Development. *ACM Multimedia* (2017).

[14] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. Multimodal deep learning for robust rgb-d object recognition. In *2015 IEEE/RSJ International meeting on Intelligent Robots and Systems (IROS)* (Sep. 2015), pp. 681–687.

[15] Fang, H., Xie, S., Tai, Y., and Lu, C. Rmpe: Regional multi-person pose estimation. In *2017 IEEE International meeting on Computer Vision (ICCV)* (Oct 2017), pp. 2353–2362.

[16] Felzenszwalb, P. F., and Huttenlocher, D. P. Pictorial structures for object recognition. *International journal of computer vision 61*, 1 (2005), 55–79.

[17] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks. *J. Mach. Learn. Res. 17*, 1 (Jan. 2016), 2096–2030.

[18] Giancola, S., Valenti, M., and Sala, R. *Metrological Qualification of the Orbbec Astra S^{TM} Structured-Light Camera*. Springer International Publishing, Cham, 2018, pp. 61–69.

[19] Girardeau-Montaut, D. Cloud compare. `https://www.danielgm.net/cc/`. Accessed: 17.10.2019.

[20] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research 9* (2010), 249–256. Cited By :2306.

[21] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] Grest, D., Woetzel, J., and Koch, R. Nonlinear body pose estimation from depth images. In *Joint Pattern Recognition Symposium* (2005), Springer, pp. 285–292.

[23] Haque, A., Peng, B., Luo, Z., Alahi, A., Yeung, S., and Fei-Fei, L. Towards viewpoint invariant 3d human pose estimation. In *ECCV* (2016).

[24] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[25] Iqbal, U., and Gall, J. Multi-person pose estimation with local joint-to-person associations. In *ECCV Workshops* (2016).

[26] Kohli, P., and Shotton, J. Key developments in human pose estimation for kinect. In *Consumer Depth Cameras for Computer Vision* (2013).

[27] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014* (Cham, 2014), D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Springer International Publishing, pp. 740–755.

[28] Martínez-González, A., Villamizar, M., Canévet, O., and Odobez, J.-M. Investigating depth domain adaptation for efficient human pose estimation. In *Computer Vision – ECCV 2018 Workshops* (Cham, 2019), L. Leal-Taixé and S. Roth, Eds., Springer International Publishing, pp. 346–363.

[29] Martínez-González, A., Villamizar, M., Canévet, O., and Odobez, J.-M. Real-time convolutional networks for depth-based human pose estimation.

[30] Microsoft. Azure kinect depth camera. `https://azure.microsoft.com/en-in/services/kinect-dk/`. Accessed: 17.10.2019.

[31] Moon, G., Chang, J., and Lee, K. M. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *The IEEE meeting on Computer Vision and Pattern Recognition (CVPR)* (2018).

[32] Motion, C. G. L. Cmu motion capture data. `http://mocap.cs.cmu.edu/`. Accessed: 20.6.2019.

[33] Patricia, N., Cariucci, F. M., and Caputo, B. Deep depth domain adaptation: A case study. *2017 IEEE International meeting on Computer Vision Workshops (ICCVW)* (2017), 2645–2650.

[34] Poppe, R. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding 108*, 1 (2007), 4 – 18. Special Issue on Vision for Human-Computer Interaction.

[35] Ruggero Ronchi, M., and Perona, P. Benchmarking and error diagnosis in multi-instance pose estimation. In *Proceedings of the IEEE International meeting on Computer Vision* (2017), pp. 369–378.

[36] Rusu, R. B., and Cousins, S. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)* (Shanghai, China, May 9-13 2011).

[37] Sarbolandi, H., Lefloch, D., and Kolb, A. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer Vision and Image Understanding 139* (2015), 1 – 20.

[38] Shakhnarovich, G., Viola, P., and Darrell, T. Fast pose estimation with parameter sensitive hashing.

[39] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from single depth images. In *CVPR 2011* (June 2011), pp. 1297–1304.

[40] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. Learning from simulated and unsupervised images through adversarial training. In *2017 IEEE meeting on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 2242–2251.

[41] Sun, M., and Savarese, S. Articulated part-based model for joint object detection and pose estimation. In *Proceedings of the 2011 International meeting on Computer Vision* (Washington, DC, USA, 2011), ICCV '11, IEEE Computer Society, pp. 723–730.

[42] Taylor, J., Shotton, J., Sharp, T., and Fitzgibbon, A. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *2012 IEEE meeting on Computer Vision and Pattern Recognition* (June 2012), pp. 103–110.

[43] Tobin, J., Fong, R. H., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International meeting on Intelligent Robots and Systems (IROS)* (2017), 23–30.

[44] Tompson, J., Jain, A., LeCun, Y., and Bregler, C. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS* (2014).

[45] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S. Training deep networks with synthetic data: Bridging the reality gap by domain randomization.

[46] Wang, K., Zhai, S., Cheng, H., Liang, X., and Lin, L. Human pose estimation from depth images via inference embedded multi-task learning. In *ACM Multimedia* (2016).

[47] Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. Convolutional pose machines. In *Proceedings of the IEEE meeting on Computer Vision and Pattern Recognition* (2016), pp. 4724–4732.

[48] Weng, L. Domain randomization for sim2real transfer. *lilianweng.github.io/lil-log* (2019).

[49] Zanuttigh, P., Dal Mutto, C., Minto, L., Marin, G., Dominio, F., and Maria Cortelazzo, G. *Time-of-flight and structured light depth cameras: Technology and applications.* 01 2016.